

## Part 1: Implement Logistic Regression

### Index

|                                     |    |
|-------------------------------------|----|
| 1] Overview                         | 3  |
| 2] Dataset Description & Analysis   | 3  |
| Age                                 | 5  |
| Blood pressure                      | 6  |
| BMI                                 | 7  |
| Diabetes Pedigree Function          | 8  |
| Glucose                             | 9  |
| Insulin                             | 10 |
| Pregnancies                         | 11 |
| SkinThickness                       | 11 |
| Analysing Correlation               | 12 |
| Treating Outliers and Non-Normality | 12 |
| Data Splitting                      | 13 |
| Designing Logistic Regression       | 13 |
| Stochastic Gradient Descent         | 13 |
| Hyperparameters                     | 14 |
| Result                              | 14 |

### List of Figures

|                            |   |
|----------------------------|---|
| Fig 1: Diabetic VS Healthy | 4 |
|----------------------------|---|

|  |    |
|--|----|
| Fig 2: Independent Variable - Age                        | 5  |
| Fig 3: Independent Variable - Blood Pressure             | 6  |
| Fig 4: Violin Plot for Blood pressure                    | 6  |
| Fig 5: Replacing zeros for Blood Pressure                | 7  |
| Fig 6: Independent Variable - BMI                        | 7  |
| Fig 7: Independent Variable - Diabetes Pedigree Function | 8  |
| Fig 8: Independent Variable - Glucose                    | 9  |
| Fig 10: Independent Variable - Insulin                   | 10 |
| Fig 11: Independent Variable - Pregnancies               | 11 |
| Fig 12: Pearson's Correlation                            | 12 |
| Fig 13: Data Normalization                               | 12 |
| Fig 14: Stochastic Gradient Descent Implementation       | 14 |
| Fig 15: Prediction using Sigmoid Function                | 14 |
| Fig 16: Logistic Regression Model scores & Accuracy      | 14 |

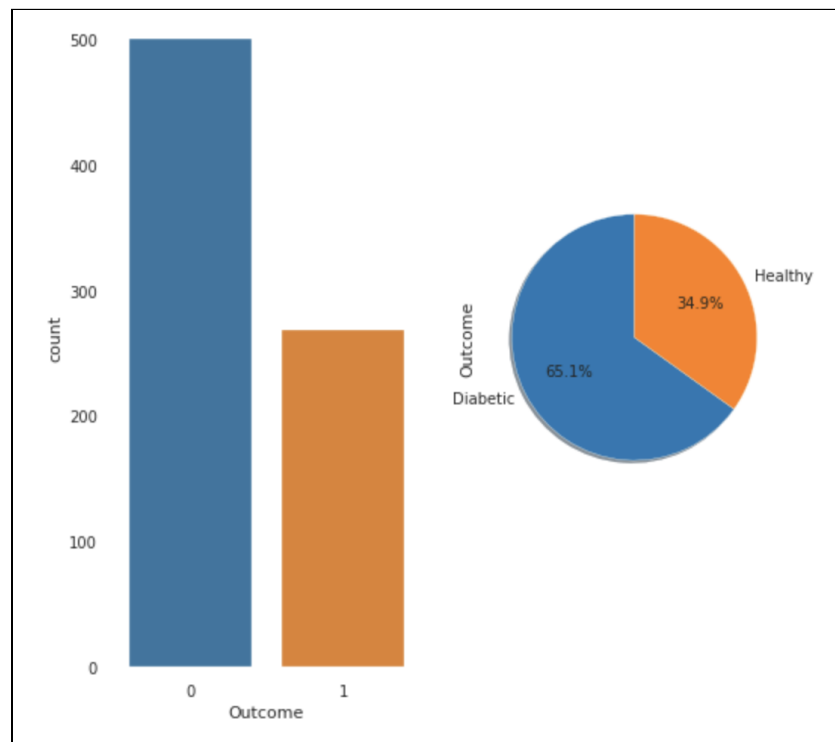
## 1] Overview:

Goal of the project is to use Gradient Descent for logistic regression to train a model using a group of hyper-parameters to classify whether a patient has diabetes(class 1) or not (class 0). Based on Pima Indians diabetes dataset.

## 2] Dataset Description & Analysis:

Pima Indians diabetes dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and can be used to predict whether a patient has diabetes based on certain diagnostic factors. Starting off, I use Python 3.3 to implement the model. The dataset has only 768 observations and 9 columns.

I have used Pandas Profiling for Exploratory Data analysis[EDA], Pandas profiling is an efficient way to get an overall as well as in-depth information about the dataset and the variables in it.



*Fig 1: Diabetic VS Healthy*

The Above figure shows that in the dataset there are 65.1% diabetic people while the other 34.9 are healthy. Since this result was an outcome based on the 8 features, I have further analysed each attribute in detail for doing feature selection.

## Age:

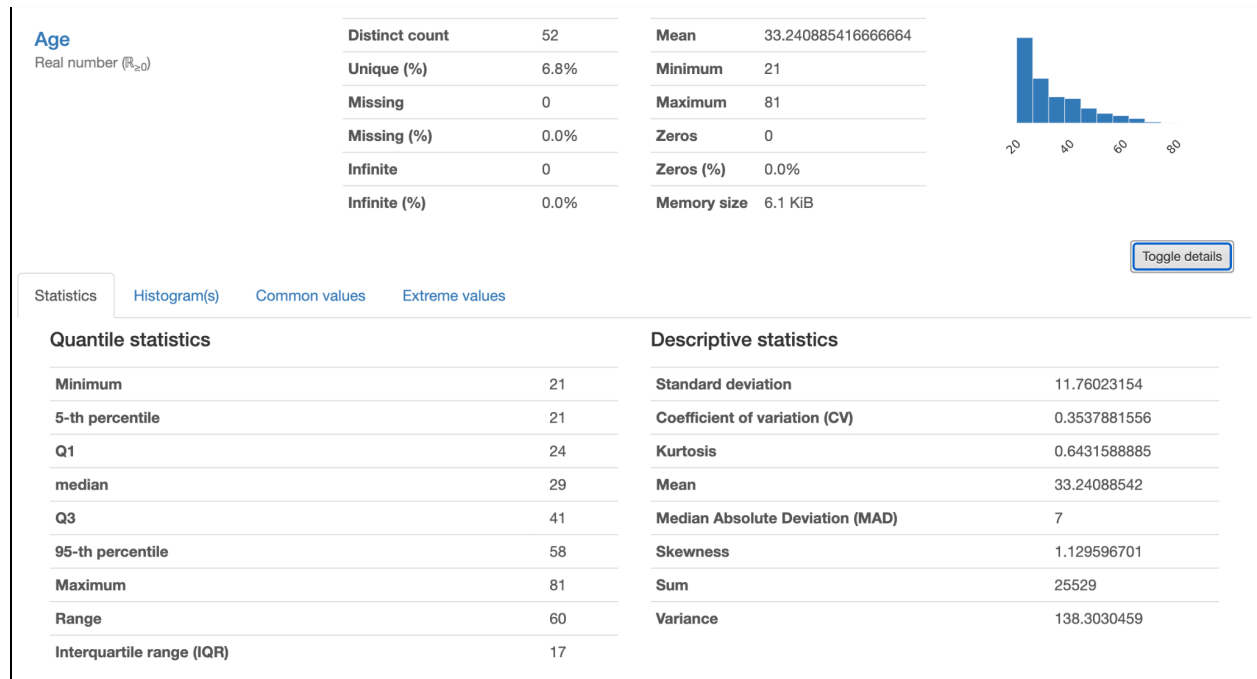


Fig 2: Independent Variable - Age

- The variable does not follow a normal distribution as it is skewed to the right. The average age is 33 whereas the median age is 29. This says that the analysis is correct since in case of normal distribution, the mean should be approximately equal to the median.
- The minimum age of 21 and the maximum age of 81 says that all the values are normal and there is no noisy data.

## Blood pressure:

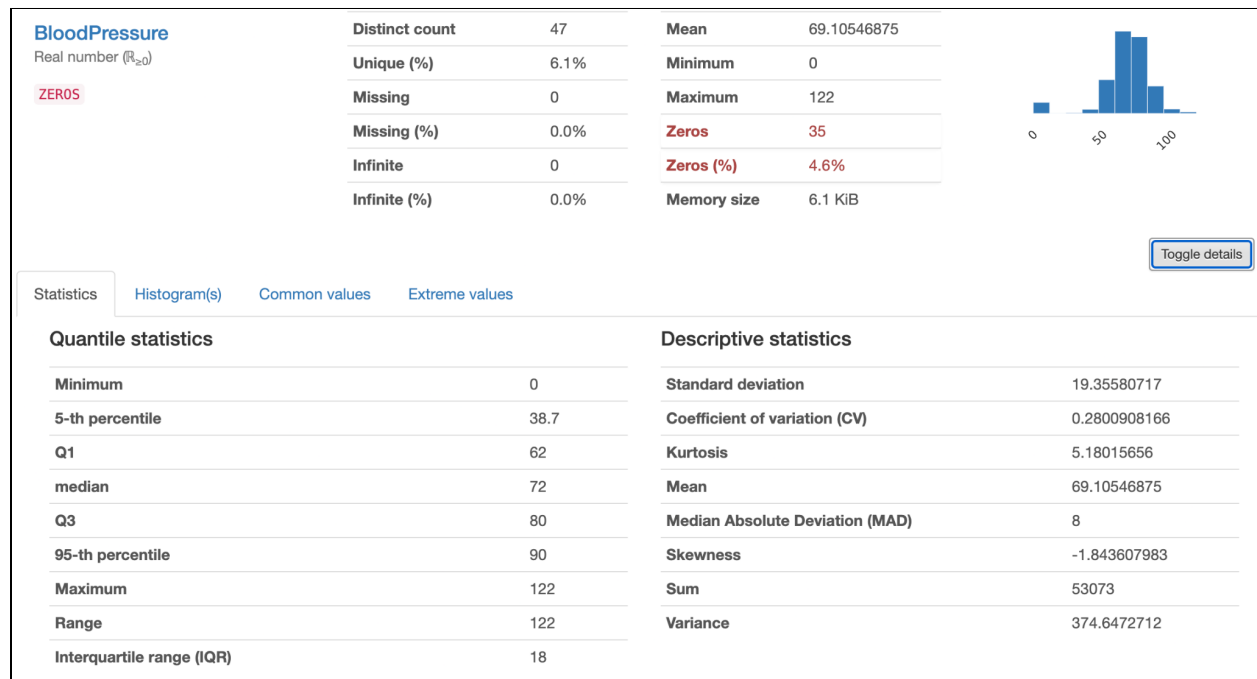


Fig 3: Independent Variable - Blood Pressure

- From the above figure I observed the minimum value for Blood Pressure is 0 (which is not possible). Therefore I replaced such values with median.

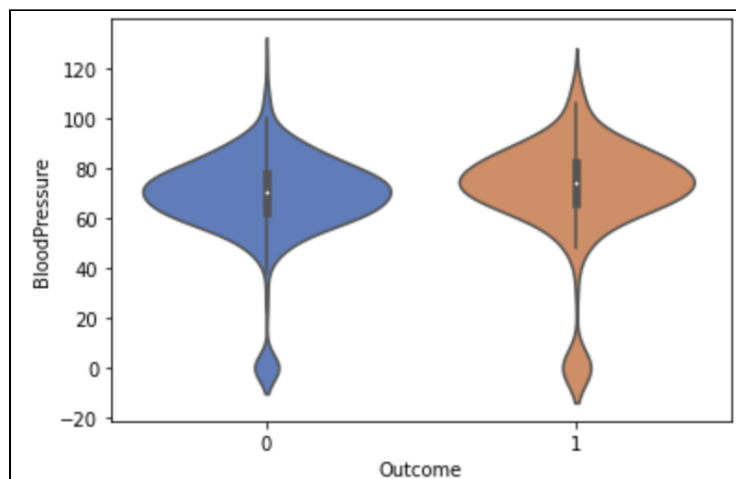


Fig 4: Violin Plot for Blood pressure

- The above graph is a violin plot for Blood Pressure from which I could see that the box plot for 1 (Diabetic) inside the violin is a little more away from the horizontal axis than the box plot for 0 (Non Diabetic), Which implied that diabetics seem to have a higher blood pressure than the

non-diabetics. And also that I need to replace all the values that are at the bottom tail of the violin indicating 0 BP.

```
#Replacing the zero-values for Blood Pressure

df1 = diabetesDF.loc[diabetesDF['Outcome'] == 1]
df2 = diabetesDF.loc[diabetesDF['Outcome'] == 0]
df1 = df1.replace({'BloodPressure':0}, np.median(df1['BloodPressure']))
df2 = df2.replace({'BloodPressure':0}, np.median(df2['BloodPressure']))
dataframe = [df1, df2]
diabetesDF = pd.concat(dataframe)
```

Fig 5: Replacing zeros for Blood Pressure

## BMI:

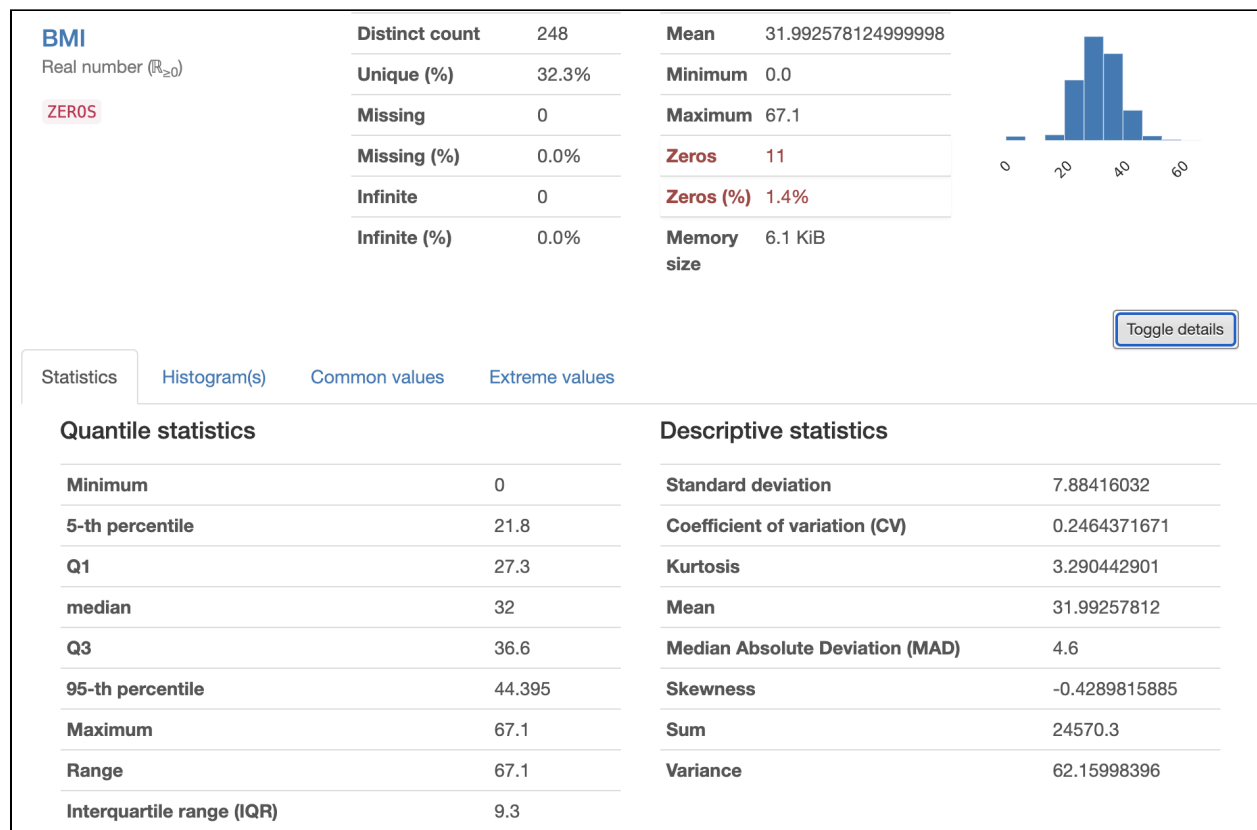


Fig 6: Independent Variable - BMI

- BMI appeared to be closely following the normal distribution as the mean and median are approximately equal. But It also has a minimum value as 0.
- The violin plot also shows that BMI for diabetics is more than BMI for non-diabetics

Diabetes Pedigree Function:

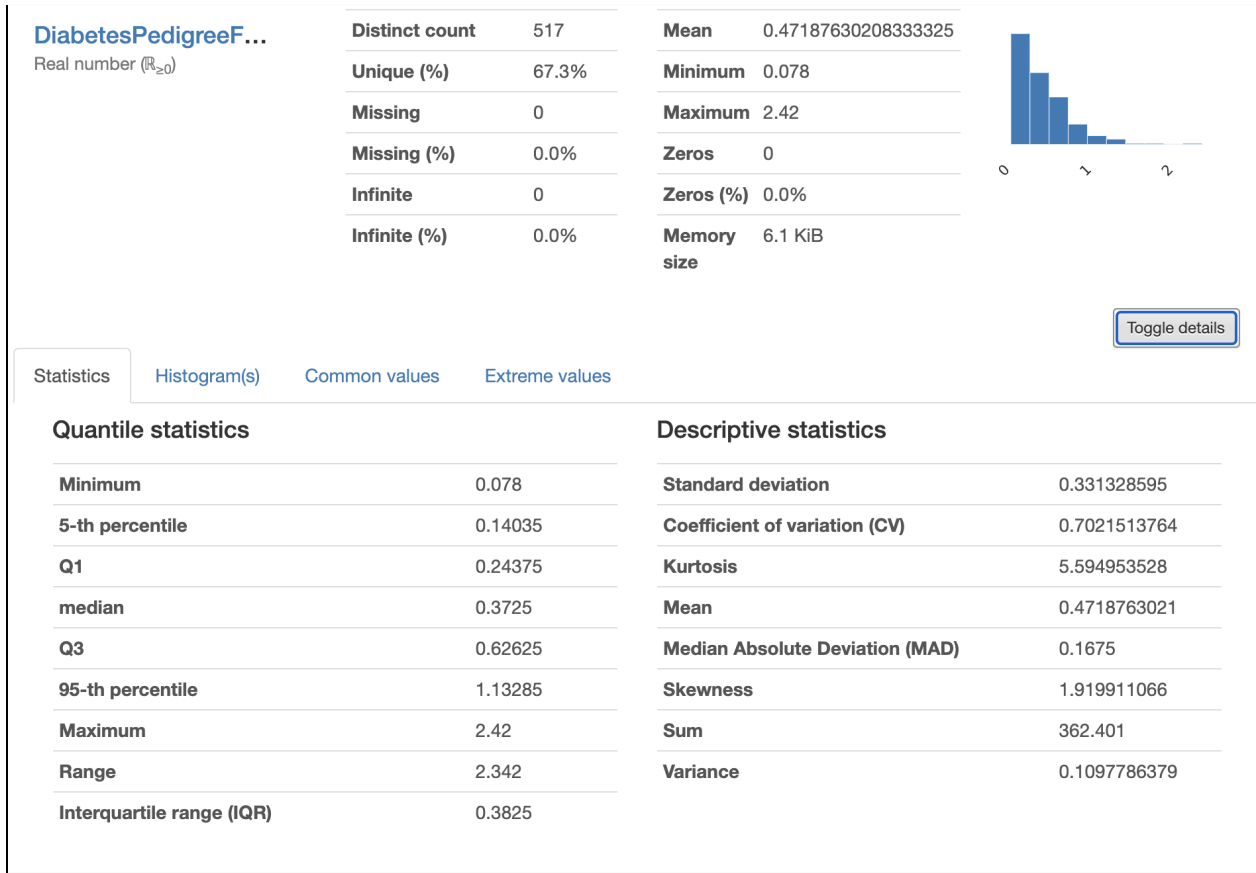


Fig 7: Independent Variable - Diabetes Pedigree Function

- Diabetes Pedigree Function is a positively skewed variable with no zero values.
- Diabetics seem to have a higher pedigree function than the non-diabetics based on violin plot.

## Glucose:

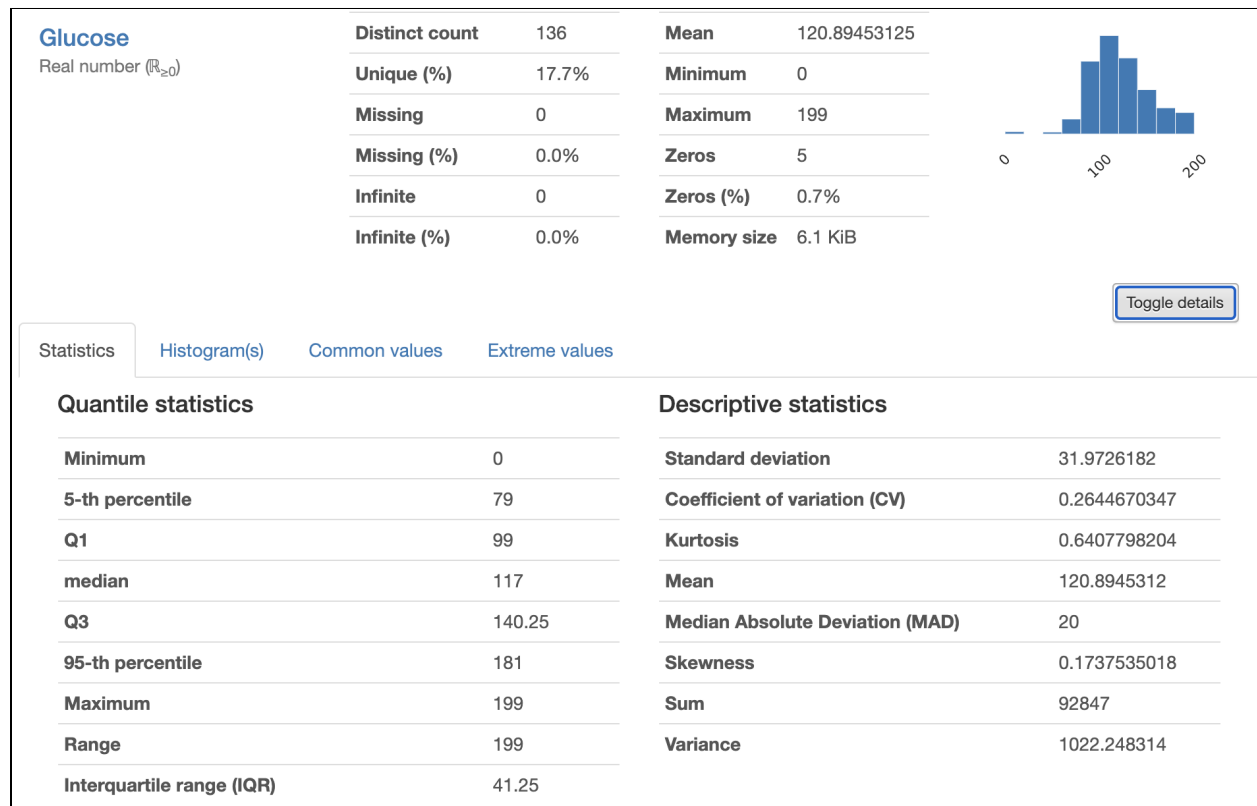


Fig 8: Independent Variable - Glucose

- Same as others, Glucose does not follow the normal distribution. We encounter zero-values in this instance as well.
- As well it contains 5 zeros, which is replaced using the same method as earlier

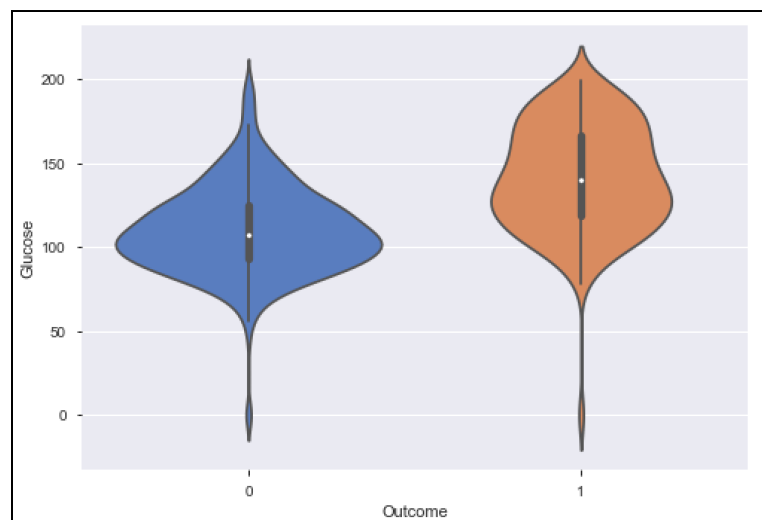


Fig 9: Violin Plot for Glucose



- Violin plot shows a massive vertical distance between the box-plot for Diabetics and Non-Diabetics, indicating that Glucose is a very important variable in model-building.

## Insulin:

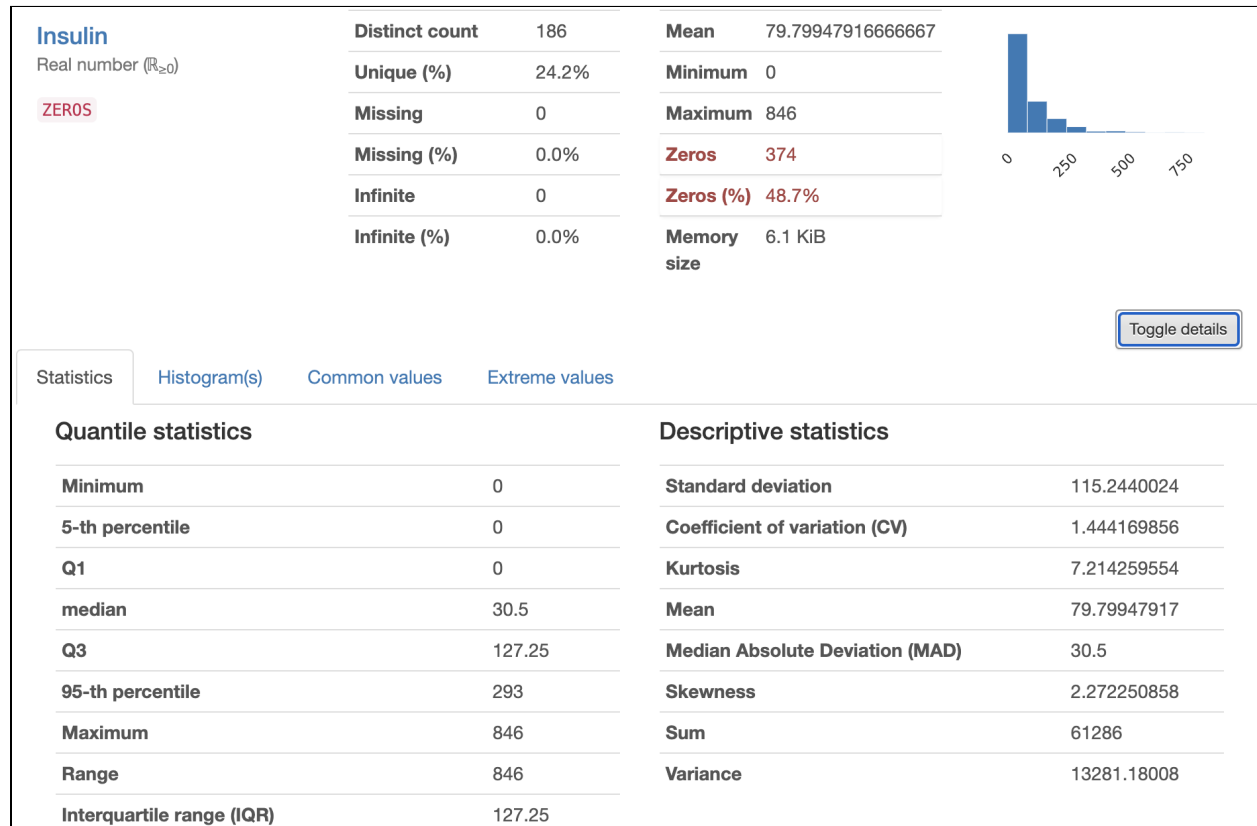


Fig 10: Independent Variable - Insulin

- As mentioned in the data dictionary available with the dataset, Insulin is the 2-Hour serum insulin ( $\mu$ U/ml).
- The EDA shows the variable is positively skewed.
- The occurrence of zero-values is high in this case, making up 48.7% of the data. Hence it has to be replaced.
- Violin plot indicates Insulin for Diabetics is lower than Non-Diabetics.

## Pregnancies:

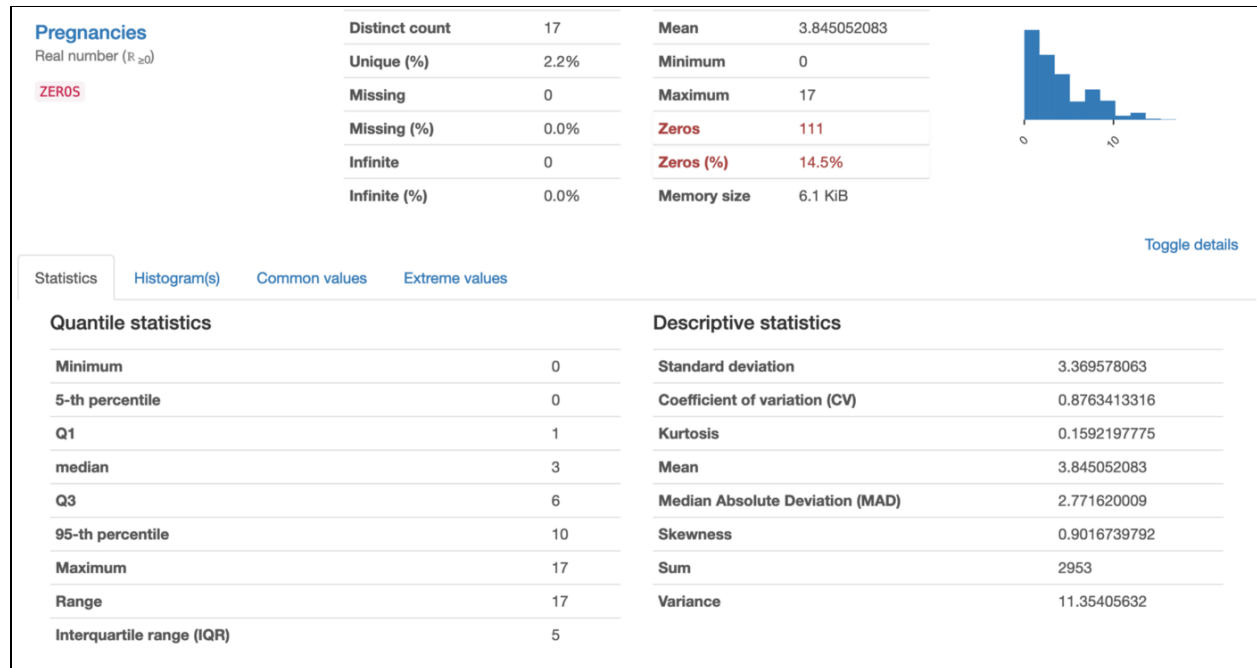


Fig 11: Independent Variable - Pregnancies

- The variable is positively skewed with 14.5% zero values.
- However the other 85.5% of Non-Zero values indicate that the study mainly includes female data only.
- Looking at the behavior using violin plots I observed diabetic women had more pregnancies than non-diabetic.

## SkinThickness:

- The data is positively skewed with 29.6% of zero values.
- Analysis also shows Skin Thickness for Diabetics is more than that of Non-Diabetics

## Analysing Correlation:

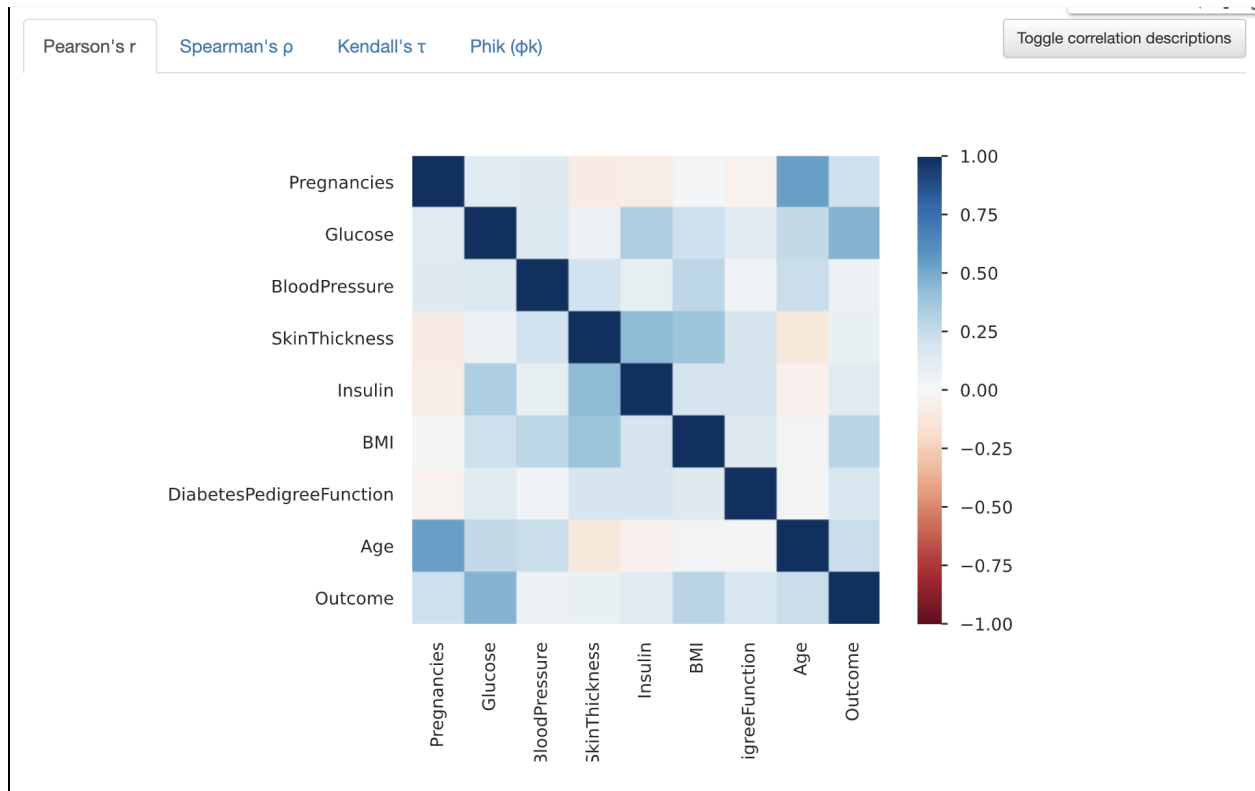


Fig 12: Pearson's Correlation

- Pearson's correlation coefficient illustrates the relationship between variables. From the above figure, a significant correlation can be observed between Pregnancies and Age.

## Treating Outliers and Non-Normality:

- After loading the dataset I first converted all the string values to numeric
- Normalized each column to values in the range of 0 to 1

```
[ ] column=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

#convert all string to float
for i in column:
    diabetesDF[i] = diabetesDF[i].astype(float)

# Rescale dataset columns to the range 0-1
scaler = MinMaxScaler()
diabetesDF = pd.DataFrame(scaler.fit_transform(diabetesDF), columns=diabetesDF.columns)
diabetesDF.head()
```

Fig 13: Data Normalization

**Data Splitting:**

- I have used k-fold cross validation to estimate the performance of the learned model on unseen data.
- I have constructed and evaluated k models and estimated the performance as the mean model performance.
- And used classification accuracy to evaluate each model.

**Designing Logistic Regression:**

- As per my research Logistic Regression uses an equation as the representation, very much like linear regression. Input values (X) are combined linearly using weights or coefficient values to predict an output value (y).
- A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value.

$$\hat{y} = 1.0 / (1.0 + e^{-(b_0 + b_1 * x_1)})$$

- Where e is the base of the Euler's number,
  - $\hat{y}$  is the predicted output,
  - $b_0$  is the bias or intercept term
  - $b_1$  is the coefficient for the single input value ( $x_1$ ).
- Each column in the data has been associated with 'b' coefficient that is learned from training data.

**Stochastic Gradient Descent:**

- As per my research Gradient Descent is the process of minimizing a function by following the gradients of the cost function.
  - This involves knowing the form of the cost as well as the derivative so that from a given point you know the gradient and can move in that direction.
- I have used stochastic gradient descent to minimize the error of the model on training data since it updates the coefficients every iteration .
  - So the way my model works is that each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated and the model is updated in order to reduce the error for the next prediction.
  - After each iteration, the coefficients (b) is updated

```

# Estimate logistic regression coefficients using stochastic gradient descent
def stochastic_gradient_descent(train, learning_rate, epochs):
    Coeff = [0.0 for i in range(len(train.iloc[0]))]
    for epoch in range(epochs):
        sum_of_error = 0
        for (idx, rows) in train.iterrows():
            row=[rows.Pregnancies,rows.Glucose,rows.BloodPressure,rows.SkinThickness,rows.Insulin,rows.BMI,rows.DiabetesPedigreeFunction,rows.Age,rows.Outcome]
            y_out = predict(row, Coeff)
            error = row[-1] - y_out
            sum_of_error += error**2
            Coeff[0] = Coeff[0] + learning_rate * error * y_out * (1.0 - y_out)
            for i in range(len(row)-1):
                Coeff[i + 1] = Coeff[i + 1] + learning_rate * error * y_out * (1.0 - y_out) * row[i]
            print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, learning_rate, sum_of_error))
    return Coeff

```

*Fig 14: Stochastic Gradient Descent Implementation*

- I have used sigmoid function to predict the outcome for a given row of input

```

[11] # Make a prediction with coefficients - Sigmoid Function
def predict(row, coefficients):
    y_out = coefficients[0]
    for i in range(len(row)-1):
        y_out += coefficients[i + 1] * row[i]
    return 1.0 / (1.0 + exp(-y_out))

```

*Fig 15: Prediction using Sigmoid Function*

### Hyperparameters:

- A k value of 5 is used for cross-validation, giving each fold  $768/5 = 153.6$  or just over 150 records to be evaluated upon each iteration.
- A learning rate of 0.2 and 10000 training epochs are chosen after a little experimentation.

### Result:

- Model gives an accuracy of about 77.5%

Scores: [73.8562091503268, 78.43137254901961, 81.69934640522875, 75.81699346405229, 75.81699346405229]  
Mean Accuracy: 77.124%

*Fig 16: Model scores & Accuracy*

## **Part 2: Implement Neural Networks**

### **Index**

|   |           |
|---|-----------|
| <b>1] Dataset Preprocessing &amp; Splitting</b> | <b>15</b> |
| <b>2] Neural Network Architecture</b>           | <b>15</b> |
| Activation Function                             | 15        |
| Loss or Cost Function                           | 15        |
| Optimizer                                       | 15        |
| Metrics   | 16        |
| Model   | 16        |
| Result & Comparison                             | 16        |

### **List of Figures**

|   |           |
|---|-----------|
| <b>Fig 17: Neural Network Preprocessing</b>             | <b>15</b> |
| <b>Fig 18: Neural Network Model Loss &amp; Accuracy</b> | <b>16</b> |
| <b>Fig 19: Model Loss &amp; Epoch</b>                   | <b>17</b> |
| <b>Fig 20: Regularization methods Comparison table</b>  | <b>17</b> |

## 1] Dataset Preprocessing & Splitting:

- I have used tensorflow to split the dataset into 70% training and 30% testing sets.
- Then I have applied one hot encoding on the outcome converting 1 into [1,0] and 0 into [0,1]

```
label = []
for lab in df_label:
    if lab == 1:
        label.append([1, 0]) # class 1
    elif lab == 0:
        label.append([0, 1]) # class 0
```

*Fig 17: Data Preprocessing*

- Then finally converted all the input data into numpy array to feed in to the model for training

## 2] Neural Network Architecture:

Input Layer: 500 nodes - Sigmoid Activation, L2 Regularization with Learning Rate(lr) of 0.00001  
Dropout Layer 1: Working at the rate of 0.2  
Hidden Layer 1: 100 nodes - Sigmoid Activation, L2 Regularization with Learning Rate(lr) of 0.00001  
Dropout Layer 2: Working at the rate of 0.2  
Output Layer: 2 node - Softmax activation, L2 Regularization with Learning Rate(lr) of 0.00001

This Neural Network is feedforward and fully connected. Every node in a given layer will be connected to all nodes in the next layer.

- **Activation Function:**

- The Sigmoid activation function has been used for all nodes except those in the output layer, which is yielding either 1 or 0.
- The output layer uses Softmax activation to yield a value between 0 to 1, which would be the predicted chance of diagnosis.

- **Loss or Cost Function:**

- Mean Squared Error loss function has been used because the output contains either 1 for positive diagnosis or 0 for negative diagnosis. MSE measures the average of the squares of the errors, that is the average squared difference between the estimated values and the actual value.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error  
n = number of data points  
Y<sub>i</sub> = observed values  
Y<sup>^</sup><sub>i</sub> = predicted values

- **Optimizer:**

- Adam has been used as the optimizer because it is an efficient way to apply gradient descent. Gradient descent is an optimization algorithm used to find the values of the parameters of a function that minimizes a loss function, in this case the mean squared error.
- Learning Rate is 0.00001

- **Metrics:**

- Tensorflow accuracy metrics is used to evaluate the model

- **Model:**

- The model is trained for 5000 epochs
- With a batch size of 10

- **Result & Comparison:**

```
Evaluate on test data
2/2 [=====] - 0s 5ms/step - loss: 0.1698 -
accuracy: 0.7806
Accuracy : 78.05627679824829 %
```

*Fig 18: Model Loss & Accuracy*



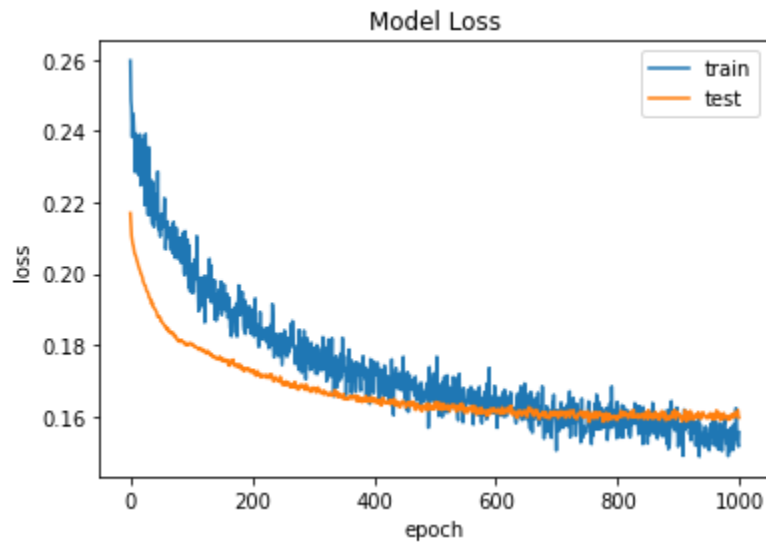


Fig 19: Model Loss & Epoch

| Network Architecture | L1 regularization % Accuracy | L2 regularization Accuracy% | Dropout %Accuracy |
|----------------------|------------------------------|-----------------------------|-------------------|
| 10-12-2              | 69.9878%                     | 65.9667%                    | 64.9982%          |
| 8-12-2               | 70.0574%                     | 63.8747%                    | 68.0781%          |
| 25-10-2              | 49.1345%                     | 66.9447%                    | 62.1034%          |
| 500-100-2            | 72.3649%                     | 75.0667%                    | 78.7803%          |
| 80-400-2             | 59.6473%                     | 71.6911%                    | 65.8867%          |
| 300-120-2            | 65.1278%                     | 68.0778%                    | 73.4623%          |
| 100-300-2            | 73.7618%                     | 63.0643%                    | 59.9867%          |

Fig 20: Regularization methods Comparison table